Learning Dense Representations of Phrases at Scale

Jinhyuk Lee[1,2*] **Mujeen Sung**[1] **Jaewoo Kang**[1] **Danqi Chen**[2]
Korea University[1] Princeton University[2]
{jinhyuk_lee,mujeensung,kangj}@korea.ac.kr
danqic@cs.princeton.edu

Abstract

Open-domain question answering can be reformulated as a phrase retrieval problem, without the need for processing documents on-demand during inference (Seo et al., 2019). However, current phrase retrieval models heavily depend on sparse representations and still underperform retriever-reader approaches. In this work, we show for the first time that we can learn *dense representations of phrases* alone that achieve much stronger performance in open-domain QA. We present an effective method to learn phrase representations from the supervision of reading comprehension tasks, coupled with novel negative sampling methods. We also propose a query-side fine-tuning strategy, which can support transfer learning and reduce the discrepancy between training and inference. On five popular open-domain QA datasets, our model *DensePhrases* improves over previous phrase retrieval models by 15%–25% absolute accuracy and matches the performance of state-of-the-art retriever-reader models. Our model is easy to parallelize due to pure dense representations and processes more than 10 questions per second on CPUs. Finally, we directly use our pre-indexed dense phrase representations for two slot filling tasks, showing the promise of utilizing DensePhrases as a dense knowledge base for downstream tasks.[1]

1 Introduction

Open-domain question answering (QA) aims to provide answers to natural-language questions using a large text corpus (Voorhees et al., 1999; Ferrucci et al., 2010; Chen and Yih, 2020). While a dominating approach is a two-stage retriever-reader approach (Chen et al., 2017; Lee et al., 2019; Guu et al., 2020; Karpukhin et al., 2020), we focus on a recent new paradigm solely based on *phrase retrieval* (Seo et al., 2019; Lee et al., 2020). Phrase retrieval highlights the use of phrase representations and finds answers purely based on the similarity search in the vector space of phrases.[2] Without relying on an expensive reader model for processing text passages, it has demonstrated great runtime efficiency at inference time.

Despite great promise, it remains a formidable challenge to build vector representations for every single phrase in a large corpus. Since phrase representations are decomposed from question representations, they are inherently less expressive than cross-attention models (Devlin et al., 2019). Moreover, the approach requires retrieving answers correctly out of billions of phrases (e.g., 6×10^{10} phrases in English Wikipedia), making the scale of the learning problem difficult. Consequently, existing approaches heavily rely on sparse representations for locating relevant documents and paragraphs while still falling behind retriever-reader models (Seo et al., 2019; Lee et al., 2020).

In this work, we investigate whether we can build fully dense phrase representations at scale for open-domain QA. First, we aim to learn strong phrase representations from the supervision of reading comprehension tasks. We propose to use data augmentation and knowledge distillation to learn better phrase representations within a single passage. We then adopt negative sampling strategies such as in-batch negatives (Henderson et al., 2017; Karpukhin et al., 2020), to better discriminate the phrases at a larger scale. Here, we present a novel method called *pre-batch negatives*, which leverages preceding mini-batches as negative examples to compensate the need of large-batch training. Lastly, we present a query-side fine-tuning strategy that dras-

[*] Work partly done while visiting Princeton University.

[1] Our code is available at https://github.com/princeton-nlp/DensePhrases.

[2] Following previous work (Seo et al., 2018), 'phrase' denotes any contiguous segment of text up to L words (including single words), which is not necessarily a linguistic phrase.

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pages 6634–6647

Category	Model	Sparse?	Storage (GB)	#Q/sec (GPU, CPU)	NQ (Acc)	SQuAD (Acc)
Retriever-Reader	DrQA (Chen et al., 2017)	✓	26	1.8, 0.6	-	29.8
	BERTSerini (Yang et al., 2019)	✓	21	2.0, 0.4	-	**38.6**
	ORQA (Lee et al., 2019)	✗	18	8.6, 1.2	33.3	20.2
	$\text{REALM}_{\text{News}}$ (Guu et al., 2020)	✗	18	8.4, 1.2	40.4	-
	DPR-multi (Karpukhin et al., 2020)	✗	76	0.9, 0.04	**41.5**	24.1
Phrase Retrieval	DenSPI (Seo et al., 2019)	✓	1,200	2.9, 2.4	8.1	36.2
	DenSPI + Sparc (Lee et al., 2020)	✓	1,547	2.1, 1.7	14.5	**40.7**
	DensePhrases (Ours)	✗	320	20.6, 13.6	**40.9**	38.0

Table 1: Retriever-reader and phrase retrieval approaches for open-domain QA. The *retriever-reader* approach retrieves a small number of relevant documents or passages from which the answers are extracted. The *phrase retrieval* approach retrieves an answer out of billions of phrase representations pre-indexed from the entire corpus. Appendix B provides detailed benchmark specification. The accuracy is measured on the test sets in the open-domain setting. NQ: Natural Questions.

tically improves phrase retrieval performance and allows for transfer learning to new domains, without re-building billions of phrase representations.

As a result, all these improvements lead to a much stronger phrase retrieval model, without the use of *any* sparse representations (Table 1). We evaluate our model, *DensePhrases*, on five standard open-domain QA datasets and achieve much better accuracies than previous phrase retrieval models (Seo et al., 2019; Lee et al., 2020), with 15%–25% absolute improvement on most datasets. Our model also matches the performance of state-of-the-art retriever-reader models (Guu et al., 2020; Karpukhin et al., 2020). Due to the removal of sparse representations and careful design choices, we further reduce the storage footprint for the full English Wikipedia from 1.5TB to 320GB, as well as drastically improve the throughput.

Finally, we envision that DensePhrases acts as a neural interface for retrieving phrase-level knowledge from a large text corpus. To showcase this possibility, we demonstrate that we can directly use DensePhrases for fact extraction, without re-building the phrase storage. With only fine-tuning the question encoder on a small number of subject-relation-object triples, we achieve state-of-the-art performance on two slot filling tasks (Petroni et al., 2021), using less than 5% of the training data.

2 Background

We first formulate the task of open-domain question answering for a set of K documents $\mathcal{D} = \{d_1, \ldots, d_K\}$. We follow the recent work (Chen et al., 2017; Lee et al., 2019) and treat all of English Wikipedia as $\mathcal{D}$, hence $K \approx 5 \times 10^6$. However, most approaches—including ours—are generic and could be applied to other collections of documents.

The task aims to provide an answer $\hat{a}$ for the input question q based on $\mathcal{D}$. In this work, we focus on the extractive QA setting, where each answer is a segment of text, or a *phrase*, that can be found in $\mathcal{D}$. Denote the set of phrases in $\mathcal{D}$ as $\mathcal{S}(\mathcal{D})$ and each phrase $s_k \in \mathcal{S}(\mathcal{D})$ consists of contiguous words $w_{\texttt{start}(k)}, \ldots, w_{\texttt{end}(k)}$ in its document $d_{\texttt{doc}(k)}$. In practice, we consider all the phrases up to $L = 20$ words in $\mathcal{D}$ and $\mathcal{S}(\mathcal{D})$ comprises a large number of 6×10^{10} phrases. An extractive QA system returns a phrase $\hat{s} = \text{argmax}_{s \in \mathcal{S}(\mathcal{D})} f(s|\mathcal{D}, q)$ where f is a scoring function. The system finally maps $\hat{s}$ to an answer string $\hat{a}$: $\texttt{TEXT}(\hat{s}) = \hat{a}$ and the evaluation is typically done by comparing the predicted answer $\hat{a}$ with a gold answer a^*.

Although we focus on the extractive QA setting, recent works propose to use a generative model as the reader (Lewis et al., 2020; Izacard and Grave, 2021), or learn a closed-book QA model (Roberts et al., 2020), which directly predicts answers without using an external knowledge source. The extractive setting provides two advantages: first, the model directly locates the source of the answer, which is more interpretable, and second, phrase-level knowledge retrieval can be uniquely adapted to other NLP tasks as we show in §7.3.

Retriever-reader. A dominating paradigm in open-domain QA is the retriever-reader approach (Chen et al., 2017; Lee et al., 2019; Karpukhin et al., 2020), which leverages a first-stage document retriever f_{retr} and only reads top $K' \ll K$ documents with a reader model f_{read}. The scoring function $f(s \mid \mathcal{D}, q)$ is decomposed as:

$$f(s \mid \mathcal{D}, q) = f_{\text{retr}}(\{d_{j_1}, \ldots, d_{j_{K'}}\} \mid \mathcal{D}, q) \times f_{\text{read}}(s \mid \{d_{j_1}, \ldots, d_{j_{K'}}\}, q), \quad (1)$$

where $\{j_1, \ldots, j_{K'}\} \subset \{1, \ldots, K\}$ and if $s \notin \mathcal{S}(\{d_{j_1}, \ldots, d_{j_{K'}}\})$, the score will be 0. It can easily adapt to passages and sentences (Yang et al., 2019; Wang et al., 2019). However, this approach suffers from error propagation when incorrect documents are retrieved and can be slow as it usually requires running an expensive reader model on every retrieved document or passage at inference time.

Phrase retrieval. Seo et al. (2019) introduce the phrase retrieval approach that encodes phrase and question representations *independently* and performs similarity search over the phrase representations to find an answer. Their scoring function f is computed as follows:

$$f(s \mid \mathcal{D}, q) = E_s(s, \mathcal{D})^\top E_q(q), \quad (2)$$

where E_s and E_q denote the phrase encoder and the question encoder respectively. As $E_s(\cdot)$ and $E_q(\cdot)$ representations are decomposable, it can support maximum inner product search (MIPS) and improve the efficiency of open-domain QA models. Previous approaches (Seo et al., 2019; Lee et al., 2020) leverage both dense and sparse vectors for phrase and question representations by taking their concatenation: $E_s(s, \mathcal{D}) = [E_{\text{sparse}}(s, \mathcal{D}), E_{\text{dense}}(s, \mathcal{D})]$.[3] However, since the sparse vectors are difficult to parallelize with dense vectors, their method essentially conducts sparse and dense vector search separately. The goal of this work is to only use dense representations, i.e., $E_s(s, \mathcal{D}) = E_{\text{dense}}(s, \mathcal{D})$, which can model $f(s \mid \mathcal{D}, q)$ solely with MIPS, as well as close the gap in performance.

3 DensePhrases

3.1 Overview

We introduce DensePhrases, a phrase retrieval model that is built on fully dense representations. Our goal is to learn a phrase encoder as well as a question encoder, so we can pre-index all the possible phrases in $\mathcal{D}$, and efficiently retrieve phrases for any question through MIPS at testing time. We outline our approach as follows:

- We first learn a high-quality phrase encoder and an (initial) question encoder from the supervision of reading comprehension tasks (§4.1), as well as incorporating effective negative sampling to better discriminate phrases at scale (§4.2, §4.3).
- Then, we fix the phrase encoder and encode all the phrases $s \in \mathcal{S}(\mathcal{D})$ and store the phrase indexing offline to enable efficient search (§5).
- Finally, we introduce an additional strategy called query-side fine-tuning (§6) by further updating the question encoder.[4] We find this step to be very effective, as it can reduce the discrepancy between training (the first step) and inference, as well as support transfer learning to new domains.

Before we present the approach in detail, we first describe our base architecture below.

3.2 Base Architecture

Our base architecture consists of a phrase encoder E_s and a question encoder E_q. Given a passage $p = w_1, \ldots, w_m$, we denote all the phrases up to L tokens as $\mathcal{S}(p)$. Each phrase s_k has start and end indicies $\texttt{start}(k)$ and $\texttt{end}(k)$ and the gold phrase is $s^* \in \mathcal{S}(p)$. Following previous work on phrase or span representations (Lee et al., 2017; Seo et al., 2018), we first apply a pre-trained language model $\mathcal{M}_p$ to obtain contextualized word representations for each passage token: $\mathbf{h}_1, \ldots, \mathbf{h}_m \in \mathbb{R}^d$. Then, we can represent each phrase $s_k \in \mathcal{S}(p)$ as the concatenation of corresponding start and end vectors:

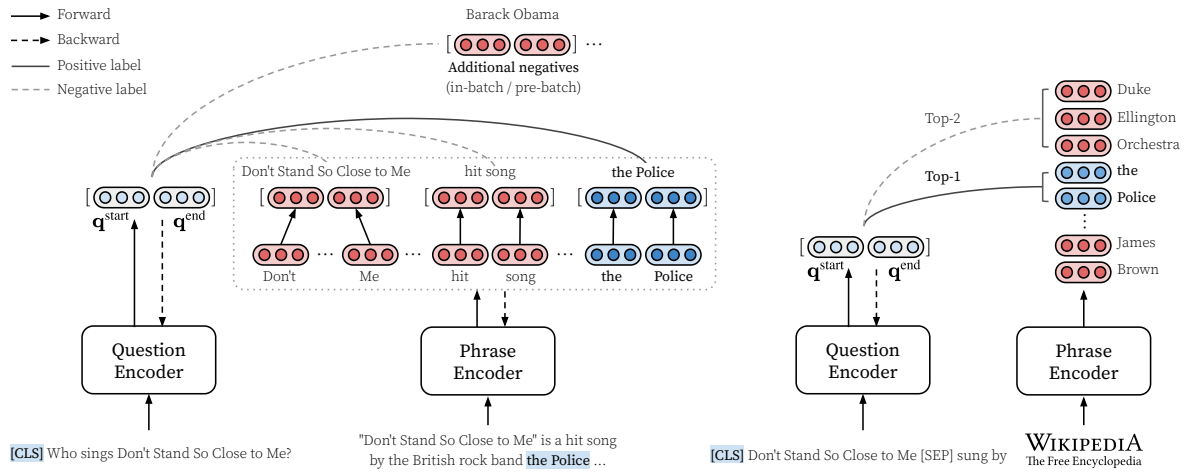
$$E_s(s_k, p) = [\mathbf{h}_{\texttt{start}(k)}, \mathbf{h}_{\texttt{end}(k)}] \in \mathbb{R}^{2d}. \quad (3)$$

A great advantage of this representation is that we eventually only need to index and store all the word vectors (we use $\mathcal{W}(\mathcal{D})$ to denote all the words in $\mathcal{D}$), instead of all the phrases $\mathcal{S}(\mathcal{D})$, which is at least one magnitude order smaller.

Similarly, we need to learn a question encoder $E_q(\cdot)$ that maps a question $q = \tilde{w}_1, \ldots, \tilde{w}_n$ to a vector of the same dimension as $E_s(\cdot)$. Since the start and end representations of phrases are produced by the same language model, we use another two different pre-trained encoders $\mathcal{M}_{q,\text{start}}$ and $\mathcal{M}_{q,\text{end}}$ to differentiate the start and end positions. We apply $\mathcal{M}_{q,\text{start}}$ and $\mathcal{M}_{q,\text{end}}$ on q separately and obtain representations $\mathbf{q}^{\text{start}}$ and $\mathbf{q}^{\text{end}}$

[3] Seo et al. (2019) use sparse representations of both paragraphs and documents and Lee et al. (2020) use contextualized sparse representations conditioned on the phrase.

[4] In this paper, we use the term *question* and *query* interchangeably as our question encoder can be naturally extended to "unnatural" queries.



(a) Single-passage training w/ additional negatives

(b) Query-side fine-tuning & Inference

Figure 1: An overview of DensePhrases. (a) We learn dense phrase representations in a single passage (§4.1) along with in-batch and pre-batch negatives (§4.2, §4.3). (b) With the top-k retrieved phrase representations from the entire text corpus (§5), we further perform query-side fine-tuning to optimize the question encoder (§6). During inference, our model simply returns the top-1 prediction.

taken from the `[CLS]` token representations respectively. Finally, $E_q(\cdot)$ simply takes their concatenation:

$$E_q(q) = [\mathbf{q}^{\text{start}}, \mathbf{q}^{\text{end}}] \in \mathbb{R}^{2d}. \quad (4)$$

Note that we use pre-trained language models to initialize $\mathcal{M}_p$, $\mathcal{M}_{q,\text{start}}$ and $\mathcal{M}_{q,\text{end}}$ and they are fine-tuned with the objectives that we will define later. In our pilot experiments, we found that SpanBERT (Joshi et al., 2020) leads to superior performance compared to BERT (Devlin et al., 2019). SpanBERT is designed to predict the information in the entire span from its two endpoints, therefore it is well suited for our phrase representations. In our final model, we use SpanBERT-base-cased as our base LMs for E_s and E_q, and hence $d = 768$.[5] See Table 5 for an ablation study.

4 Learning Phrase Representations

In this section, we start by learning dense phrase representations from the supervision of reading comprehension tasks, i.e., a single passage p contains an answer a^* to a question q. Our goal is to learn strong dense representations of phrases for $s \in \mathcal{S}(p)$, which can be retrieved by a dense representation of the question and serve as a direct answer (§4.1). Then, we introduce two different negative sampling methods (§4.2, §4.3), which encourage the phrase representations to be better discriminated at the full Wikipedia scale. See Figure 1 for an overview of DensePhrases.

4.1 Single-passage Training

To learn phrase representations in a single passage along with question representations, we first maximize the log-likelihood of the start and end positions of the gold phrase s^* where $\texttt{TEXT}(s^*) = a^*$. The training loss for predicting the start position of a phrase given a question is computed as:

$$\begin{aligned} z_1^{\text{start}}, \ldots, z_m^{\text{start}} &= [\mathbf{h}_1^\top \mathbf{q}^{\text{start}}, \ldots, \mathbf{h}_m^\top \mathbf{q}^{\text{start}}], \\ P^{\text{start}} &= \text{softmax}(z_1^{\text{start}}, \ldots, z_m^{\text{start}}), \\ \mathcal{L}_{\text{start}} &= -\log P^{\text{start}}_{\texttt{start}(s^*)}. \end{aligned} \quad (5)$$

We can define $\mathcal{L}_{\text{end}}$ in a similar way and the final loss for the single-passage training is

$$\mathcal{L}_{\text{single}} = \frac{\mathcal{L}_{\text{start}} + \mathcal{L}_{\text{end}}}{2}. \quad (6)$$

This essentially learns reading comprehension without any cross-attention between the passage and the question tokens, which fully decomposes phrase and question representations.

Data augmentation Since the contextualized word representations $\mathbf{h}_1, \ldots, \mathbf{h}_m$ are encoded in a query-agnostic way, they are always inferior to

[5] Our base model is largely inspired by DenSPI (Seo et al., 2019), although we deviate from theirs as follows. (1) We remove coherency scalars and don't split any vectors. (2) DenSPI uses a shared encoder for phrases and questions while we use 3 separate language models initialized from the same pre-trained model. (3) We use SpanBERT instead of BERT.

query-dependent representations in cross-attention models (Devlin et al., 2019), where passages are fed along with the questions concatenated by a special token such as `[SEP]`. We hypothesize that one key reason for the performance gap is that reading comprehension datasets only provide a few annotated questions in each passage, compared to the set of possible answer phrases. Learning from this supervision is not easy to differentiate similar phrases in one passage (e.g., s^* = *Charles, Prince of Wales* and another s = *Prince George* for a question q = *Who is next in line to be the monarch of England?*).

Following this intuition, we propose to use a simple model to generate additional questions for data augmentation, based on a T5-large model (Raffel et al., 2020). To train the question generation model, we feed a passage p with the gold answer s^* highlighted by inserting surrounding special tags. Then, the model is trained to maximize the log-likelihood of the question words of q. After training, we extract all the named entities in each training passage as candidate answers and feed the passage p with each candidate answer to generate questions. We keep the question-answer pairs only when a cross-attention reading comprehension model[6] makes a correct prediction on the generated pair. The remaining generated QA pairs $\{(\bar{q}_1, \bar{s}_1), (\bar{q}_2, \bar{s}_2), \ldots, (\bar{q}_r, \bar{s}_r)\}$ are directly augmented to the original training set.

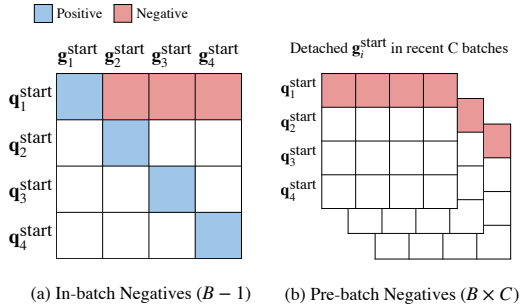
Distillation We also propose improving the phrase representations by distilling knowledge from a cross-attention model (Hinton et al., 2015). We minimize the Kullback–Leibler divergence between the probability distribution from our phrase encoder and that from a standard SpanBERT-base QA model. The loss is computed as follows:

$$\mathcal{L}_{\text{distill}} = \frac{\text{KL}(P^{\text{start}} || P_c^{\text{start}}) + \text{KL}(P^{\text{end}} || P_c^{\text{end}})}{2}, \tag{7}$$

where P^{start} (and P^{end}) is defined in Eq. (5) and P_c^{start} and P_c^{end} denote the probability distributions used to predict the start and end positions of answers in the cross-attention model.

4.2 In-batch Negatives

Eventually, we need to build phrase representations for billions of phrases. Therefore, a bigger challenge is to incorporate more phrases as negatives so the representations can be better discriminated at a larger scale. While Seo et al. (2019) simply sample two negative passages based on question similarity, we use in-batch negatives for our dense phrase representations, which has been shown to be effective in learning dense passage representations before (Karpukhin et al., 2020).



(a) In-batch Negatives ($B-1$) (b) Pre-batch Negatives ($B \times C$)

Figure 2: Two types of negative samples for the first batch item ($\mathbf{q}_1^{\text{start}}$) in a mini-batch of size $B = 4$ and $C = 3$. Note that the negative samples for the end representations ($\mathbf{q}_i^{\text{end}}$) are obtained in a similar manner. See §4.2 and §4.3 for more details.

As shown in Figure 2 (a), for the i-th example in a mini-batch of size B, we denote the hidden representations of the gold start and end positions $\mathbf{h}_{\texttt{start}(s^*)}$ and $\mathbf{h}_{\texttt{end}(s^*)}$ as $\mathbf{g}_i^{\text{start}}$ and $\mathbf{g}_i^{\text{end}}$, as well as the question representation as $[\mathbf{q}_i^{\text{start}}, \mathbf{q}_i^{\text{end}}]$. Let $\mathbf{G}^{\text{start}}, \mathbf{G}^{\text{end}}, \mathbf{Q}^{\text{start}}, \mathbf{Q}^{\text{end}}$ be the $B \times d$ matrices and each row corresponds to $\mathbf{g}_i^{\text{start}}, \mathbf{g}_i^{\text{end}}, \mathbf{q}_i^{\text{start}}, \mathbf{q}_i^{\text{end}}$ respectively. Basically, we can treat all the gold phrases from other passages in the same mini-batch as negative examples. We compute $\mathbf{S}^{\text{start}} = \mathbf{Q}^{\text{start}}\mathbf{G}^{\text{start}\intercal}$ and $\mathbf{S}^{\text{end}} = \mathbf{Q}^{\text{end}}\mathbf{G}^{\text{end}\intercal}$ and the i-th row of $\mathbf{S}^{\text{start}}$ and $\mathbf{S}^{\text{end}}$ return B scores each, including a positive score and $B-1$ negative scores: $s_1^{\text{start}}, \ldots, s_B^{\text{start}}$ and $s_1^{\text{end}}, \ldots, s_B^{\text{end}}$. Similar to Eq. (5), we can compute the loss function for the i-th example as:

$$\begin{aligned} P_i^{\text{start\_ib}} &= \text{softmax}(s_1^{\text{start}}, \ldots, s_B^{\text{start}}), \\ P_i^{\text{end\_ib}} &= \text{softmax}(s_1^{\text{end}}, \ldots, s_B^{\text{end}}), \\ \mathcal{L}_{\text{neg}} &= -\frac{\log P_i^{\text{start\_ib}} + \log P_i^{\text{end\_ib}}}{2}, \end{aligned} \tag{8}$$

We also attempted using non-gold phrases from other passages as negatives but did not find a meaningful improvement.

4.3 Pre-batch Negatives

The in-batch negatives usually benefit from a large batch size (Karpukhin et al., 2020). However, it is challenging to further increase batch sizes, as they are bounded by the size of GPU memory. Next, we propose a novel negative sampling method

[6] SpanBERT-large, 88.2 EM on SQuAD.

called *pre-batch negatives*, which can effectively utilize the representations from the preceding C mini-batches (Figure 2 (b)). In each iteration, we maintain a FIFO queue of C mini-batches to cache phrase representations $\mathbf{G}^{\text{start}}$ and $\mathbf{G}^{\text{end}}$. The cached phrase representations are then used as negative samples for the next iteration, providing $B \times C$ additional negative samples in total.[7]

These pre-batch negatives are used together with in-batch negatives and the training loss is the same as Eq. (8), except that the gradients are *not* back-propagated to the cached pre-batch negatives. After warming up the model with in-batch negatives, we simply shift from in-batch negatives ($B-1$ negatives) to in-batch and pre-batch negatives (hence a total number of $B \times C + B - 1$ negatives). For simplicity, we use $\mathcal{L}_{\text{neg}}$ to denote the loss for both in-batch negatives and pre-batch negatives. Since we do not retain the computational graph for pre-batch negatives, the memory consumption of pre-batch negatives is much more manageable while allowing an increase in the number of negative samples.

4.4 Training Objective

Finally, we optimize all the three losses together, on both annotated reading comprehension examples and generated questions from §4.1:

$$\mathcal{L} = \lambda_1 \mathcal{L}_{\text{single}} + \lambda_2 \mathcal{L}_{\text{distill}} + \lambda_3 \mathcal{L}_{\text{neg}}, \quad (9)$$

where $\lambda_1, \lambda_2, \lambda_3$ determine the importance of each loss term. We found that $\lambda_1 = 1$, $\lambda_2 = 2$, and $\lambda_3 = 4$ works well in practice. See Table 5 and Table 6 for an ablation study of different components.

5 Indexing and Search

Indexing After training the phrase encoder E_s, we need to encode all the phrases $\mathcal{S}(\mathcal{D})$ in the entire English Wikipedia $\mathcal{D}$ and store an index of the phrase dump. We segment each document $d_i \in \mathcal{D}$ into a set of natural paragraphs, from which we obtain token representations for each paragraph using $E_s(\cdot)$. Then, we build a phrase dump $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_{|\mathcal{W}(\mathcal{D})|}] \in \mathbb{R}^{|\mathcal{W}(\mathcal{D})| \times d}$ by stacking the token representations from all the paragraphs in $\mathcal{D}$. Note that this process is computationally expensive and takes about hundreds of GPU hours with a large disk footprint. To reduce the size of phrase dump, we follow and modify several techniques introduced in Seo et al. (2019) (see Appendix E for details). After indexing, we can use two rows i and j of $\mathbf{H}$ to represent a dense phrase representation $[\mathbf{h}_i, \mathbf{h}_j]$. We use `faiss` (Johnson et al., 2017) for building a MIPS index of $\mathbf{H}$.[8]

Search For a given question q, we can find the answer $\hat{s}$ as follows:

$$\begin{aligned} \hat{s} &= \underset{s_{(i,j)}}{\text{argmax}}\, E_s(s_{(i,j)}, \mathcal{D})^\top E_q(q), \\ &= \underset{s_{(i,j)}}{\text{argmax}} (\mathbf{H}\mathbf{q}^{\text{start}})_i + (\mathbf{H}\mathbf{q}^{\text{end}})_j, \end{aligned} \quad (10)$$

where $s_{(i,j)}$ denotes a phrase with start and end indices as i and j in the index $\mathbf{H}$. We can compute the argmax of $\mathbf{H}\mathbf{q}^{\text{start}}$ and $\mathbf{H}\mathbf{q}^{\text{end}}$ efficiently by performing MIPS over $\mathbf{H}$ with $\mathbf{q}^{\text{start}}$ and $\mathbf{q}^{\text{end}}$. In practice, we search for the top-k start and top-k end positions separately and perform a constrained search over their end and start positions respectively such that $1 \le i \le j < i + L \le |\mathcal{W}(\mathcal{D})|$.

6 Query-side Fine-tuning

So far, we have created a phrase dump $\mathbf{H}$ that supports efficient MIPS search. In this section, we propose a novel method called query-side fine-tuning by only updating the question encoder E_q to correctly retrieve a desired answer a^* for a question q given $\mathbf{H}$. Formally speaking, we optimize the marginal log-likelihood of the gold answer a^* for a question q, which resembles the weakly-supervised QA setting in previous work (Lee et al., 2019; Min et al., 2019). For every question q, we retrieve top k phrases and minimize the objective:

$$\mathcal{L}_{\text{query}} = -\log \frac{\sum_{s \in \tilde{\mathcal{S}}(q), \texttt{TEXT}(s) = a^*} \exp\left(f(s|\mathcal{D}, q)\right)}{\sum_{s \in \tilde{\mathcal{S}}(q)} \exp\left(f(s|\mathcal{D}, q)\right)}, \quad (11)$$

where $f(s|\mathcal{D}, q)$ is the score of the phrase s (Eq. (2)) and $\tilde{\mathcal{S}}(q)$ denotes the top k phrases for q (Eq. (10)). In practice, we use $k = 100$ for all the experiments.

There are several advantages for doing this: (1) we find that query-side fine-tuning can reduce the discrepancy between training and inference, and hence improve the final performance substantially (§8). Even with effective negative sampling, the model only sees a small portion of passages compared to the full scale of $\mathcal{D}$ and this training objective can effectively fill in the gap. (2) This training strategy allows for transfer learning to unseen

[7] This approach is inspired by the momentum contrast idea proposed in unsupervised visual representation learning (He et al., 2020). Contrary to their approach, we have separate encoders for phrases and questions and back-propagate to both during training without a momentum update.

[8] We use `IVFSQ4` with 1M clusters and set n-probe to 256.

domains, without rebuilding the entire phrase index. More specifically, the model is able to quickly adapt to new QA tasks (e.g., WebQuestions) when the phrase dump is built using SQuAD or Natural Questions. We also find that this can transfers to non-QA tasks when the query is written in a different format. In §7.3, we show the possibility of directly using DensePhrases for slot filling tasks by using a query such as *(Michael Jackson, is a singer of, x)*. In this regard, we can view our model as a dense knowledge base that can be accessed by many different types of queries and it is able to return phrase-level knowledge efficiently.

7 Experiments

7.1 Setup

Datasets. We use two reading comprehension datasets: SQuAD (Rajpurkar et al., 2016) and Natural Questions (NQ) (Kwiatkowski et al., 2019) to learn phrase representations, in which a single gold passage is provided for each question. For the open-domain QA experiments, we evaluate our approach on five popular open-domain QA datasets: Natural Questions, WebQuestions (WQ) (Berant et al., 2013), CuratedTREC (TREC) (Baudiš and Šedivỳ, 2015), TriviaQA (TQA) (Joshi et al., 2017), and SQuAD. Note that we only use SQuAD and/or NQ to build the phrase index and perform query-side fine-tuning (§6) for other datasets.

We also evaluate our model on two slot filling tasks, to show how to adapt our DensePhrases for other knowledge-intensive NLP tasks. We focus on using two slot filling datasets from the KILT benchmark (Petroni et al., 2021): T-REx (Elsahar et al., 2018) and zero-shot relation extraction (Levy et al., 2017). Each query is provided in the form of "{subject entity} `[SEP]` {relation}" and the answer is the object entity. Appendix C provides the statistics of all the datasets.

Implementation details. We denote the training datasets used for reading comprehension (Eq. (9)) as $\mathcal{C}_{\text{phrase}}$. For open-domain QA, we train two versions of phrase encoders, each of which are trained on $\mathcal{C}_{\text{phrase}} = \{\text{SQuAD}\}$ and $\{\text{NQ}, \text{SQuAD}\}$, respectively. We build the phrase dump $\mathbf{H}$ for the 2018-12-20 Wikipedia snapshot and perform query-side fine-tuning on each dataset using Eq. (11). For slot filling, we use the same phrase dump for open-domain QA, $\mathcal{C}_{\text{phrase}} = \{\text{NQ}, \text{SQuAD}\}$ and perform query-side fine-tuning on randomly sampled 5K or 10K training examples to see how rapidly our model adapts to the new query types. See Appendix D for details on the hyperparameters and Appendix A for an analysis of computational cost.

Model	SQuAD		NQ (Long)	
	EM	F1	EM	F1
Query-Dependent				
BERT-base	80.8	88.5	69.9	78.2
SpanBERT-base	85.7	92.2	73.2	81.0
Query-Agnostic				
DilBERT (Siblini et al., 2020)	63.0	72.0	-	-
DeFormer (Cao et al., 2020)	-	72.1	-	-
DenSPI†	73.6	81.7	68.2	76.1
DenSPI + Sparc†	76.4	84.8	-	-
DensePhrases (ours)	**78.3**	**86.3**	**71.9**	**79.6**

Table 2: Reading comprehension results, evaluated on the development sets of SQuAD and Natural Questions. Underlined numbers are estimated from the figures from the original papers. †: BERT-large model.

7.2 Experiments: Question Answering

Reading comprehension. In order to show the effectiveness of our phrase representations, we first evaluate our model in the reading comprehension setting for SQuAD and NQ and report its performance with other query-agnostic models (Eq. (9) without query-side fine-tuning). This problem was originally formulated by Seo et al. (2018) as the phrase-indexed question answering (PIQA) task.

Compared to previous query-agnostic models, our model achieves the best performance of 78.3 EM on SQuAD by improving the previous phrase retrieval model (DenSPI) by 4.7% (Table 2). Although it is still behind cross-attention models, the gap has been greatly reduced and serves as a strong starting point for the open-domain QA model.

Open-domain QA. Experimental results on open-domain QA are summarized in Table 3. Without any sparse representations, DensePhrases outperforms previous phrase retrieval models by a large margin and achieves a 15%–25% absolute improvement on all datasets except SQuAD. Training the model of Lee et al. (2020) on $\mathcal{C}_{\text{phrase}} = \{\text{NQ}, \text{SQuAD}\}$ only increases the result from 14.5% to 16.5% on NQ, demonstrating that it does not suffice to simply add more datasets for training phrase representations. Our performance is also competitive with recent retriever-reader models (Karpukhin et al., 2020), while running much faster during inference (Table 1).

Model		NQ	WQ	TREC	TQA	SQuAD
Retriever-reader	$\mathcal{C}_{\text{retr}}$: (Pre-)Training					
DrQA (Chen et al., 2017)	-	-	20.7	25.4	-	29.8
BERT + BM25 (Lee et al., 2019)	-	26.5	17.7	21.3	47.1	**33.2**
ORQA (Lee et al., 2019)	{Wiki.}†	33.3	36.4	30.1	45.0	20.2
$\text{REALM}_{\text{News}}$ (Guu et al., 2020)	{Wiki., CC-News}†	40.4	40.7	42.9	-	-
DPR-multi (Karpukhin et al., 2020)	{NQ, WQ, TREC, TQA}	**41.5**	**42.4**	**49.4**	**56.8**	24.1
Phrase retrieval	$\mathcal{C}_{\text{phrase}}$: Training					
DenSPI (Seo et al., 2019)	{SQuAD}	8.1*	11.1*	31.6*	30.7*	36.2
DenSPI + Sparc (Lee et al., 2020)	{SQuAD}	14.5*	17.3*	35.7*	34.4*	**40.7**
DenSPI + Sparc (Lee et al., 2020)	{NQ, SQuAD}	16.5	-	-	-	-
DensePhrases (ours)	{SQuAD}	31.2	36.3	50.3	**53.6**	39.4
DensePhrases (ours)	{NQ, SQuAD}	**40.9**	**37.5**	**51.0**	50.7	38.0

Table 3: Open-domain QA results. We report exact match (EM) on the test sets. We also show the additional training or pre-training datasets for learning the retriever models ($\mathcal{C}_{\text{retr}}$) and creating the phrase dump ($\mathcal{C}_{\text{phrase}}$). *: no supervision using target training data (zero-shot). †: unlabeled data used for extra pre-training.

Model	T-REx		ZsRE	
	Acc	F1	Acc	F1
DPR + BERT	-	-	4.47	27.09
DPR + BART	11.12	11.41	18.91	20.32
RAG	23.12	23.94	36.83	39.91
$\text{DensePhrases}^{5\text{K}}$	25.32	29.76	40.39	45.89
$\text{DensePhrases}^{10\text{K}}$	**27.84**	**32.34**	**41.34**	**46.79**

Table 4: Slot filling results on the test sets of T-REx and Zero shot RE (ZsRE) in the KILT benchmark. We report KILT-AC and KILT-F1 (denoted as *Acc* and *F1* in the table), which consider both span-level accuracy and correct retrieval of evidence documents.

7.3 Experiments: Slot Filling

Table 4 summarizes the results on the two slot filling datasets, along with the baseline scores provided by Petroni et al. (2021). The only extractive baseline is DPR + BERT, which performs poorly in zero-shot relation extraction. On the other hand, our model achieves competitive performance on all datasets and achieves state-of-the-art performance on two datasets using only 5K training examples.

8 Analysis

Ablation of phrase representations. Table 5 shows the ablation result of our model on SQuAD. Upon our choice of architecture, augmenting training set with generated questions (QG = ✓) and performing distillation from cross-attention models (Distill = ✓) improve performance up to EM = 78.3. We attempted adding the generated questions to the training of the SpanBERT-QA model but find a 0.3% improvement, which validates that data sparsity is a bottleneck for query-agnostic models.

Model	$\mathcal{M}$	Share	Split	QG	Distill	EM
DenSPI	Bb.	✓	✓	✗	✗	70.2
	Sb.	✓	✓	✗	✗	68.5
	Bl.	✓	✓	✗	✗	73.6
Dense	Bb.	✓	✗	✗	✗	70.2
Phrases	Bb.	✗	✗	✗	✗	71.9
	Sb.	✗	✗	✗	✗	73.2
	Sb.	✗	✗	✓	✗	76.3
	Sb.	✗	✗	✓	✓	**78.3**

Table 5: Ablation of DensePhrases on the development set of SQuAD. Bb: BERT-base, Sb: SpanBERT-base, Bl: BERT-large. Share: whether question and phrase encoders are shared or not. Split: whether the full hidden vectors are kept or split into start and end vectors. QG: question generation (§4.1). Distill: distillation (Eq.(7)). DenSPI (Seo et al., 2019) also included a coherency scalar and see their paper for more details.

Effect of batch negatives. We further evaluate the effectiveness of various negative sampling methods introduced in §4.2 and §4.3. Since it is computationally expensive to test each setting at the full Wikipedia scale, we use a smaller text corpus $\mathcal{D}_{\text{small}}$ of all the gold passages in the development sets of Natural Questions, for the ablation study. Empirically, we find that results are generally well correlated when we gradually increase the size of $|\mathcal{D}|$. As shown in Table 6, both in-batch and pre-batch negatives bring substantial improvements. While using a larger batch size ($B = 84$) is beneficial for in-batch negatives, the number of preceding batches in pre-batch negatives is optimal when $C = 2$. Surprisingly, the pre-batch negatives also improve the performance when $\mathcal{D} = \{p\}$.

Type	B	C	$\mathcal{D} = \{p\}$	$\mathcal{D} = \mathcal{D}_{\text{small}}$
None	48	-	70.4	35.3
+ In-batch	48	-	70.5	52.4
	84	-	70.3	54.2
+ Pre-batch	84	1	71.6	59.8
	84	2	**71.9**	**60.4**
	84	4	71.2	59.8

Table 6: Effect of in-batch negatives and pre-batch negatives on the development set of Natural Questions. B: batch size. C: number of preceding mini-batches used in pre-batch negatives. $\mathcal{D}_{\text{small}}$: all the gold passages in the development set of NQ. $\{p\}$: single passage.

Effect of query-side fine-tuning. We summarize the effect of query-side fine-tuning in Table 7. For the datasets that were not used for training the phrase encoders (TQA, WQ, TREC), we observe a 15% to 20% improvement after query-side fine-tuning. Even for the datasets that have been used (NQ, SQuAD), it leads to significant improvements (e.g., 32.6%→40.9% on NQ for $\mathcal{C}_{\text{phrase}}$ = {NQ}) and it clearly demonstrates it can effectively reduce the discrepancy between training and inference.

9 Related Work

Learning effective dense representations of words is a long-standing goal in NLP (Bengio et al., 2003; Collobert et al., 2011; Mikolov et al., 2013; Peters et al., 2018; Devlin et al., 2019). Beyond words, dense representations of many different granularities of text such as sentences (Le and Mikolov, 2014; Kiros et al., 2015) or documents (Yih et al., 2011) have been explored. While dense phrase representations have been also studied for statistical machine translation (Cho et al., 2014) or syntactic parsing (Socher et al., 2010), our work focuses on learning dense phrase representations for QA and any other knowledge-intensive tasks where phrases can be easily retrieved by performing MIPS.

This type of dense retrieval has been also studied for sentence and passage retrieval (Humeau et al., 2019; Karpukhin et al., 2020) (see Lin et al., 2020 for recent advances in dense retrieval). While DensePhrases is explicitly designed to retrieve phrases that can be used as an answer to given queries, retrieving phrases also naturally entails retrieving larger units of text, provided the datastore maintains the mapping between each phrase and the sentence and passage in which it occurs.

QS	NQ	WQ	TREC	TQA	SQuAD
$\mathcal{C}_{\text{phrase}}$ = {SQuAD}					
✗	12.3	11.8	36.9	34.6	35.5
✓	31.2	36.3	50.3	**53.6**	**39.4**
$\mathcal{C}_{\text{phrase}}$ = {NQ}					
✗	32.6	21.1	32.3	32.4	20.7
✓	**40.9**	37.1	49.7	49.2	25.7
$\mathcal{C}_{\text{phrase}}$ = {NQ, SQuAD}					
✗	28.9	18.9	34.9	31.9	33.2
✓	**40.9**	**37.5**	**51.0**	50.7	38.0

Table 7: Effect of query-side fine-tuning in DensePhrases on each test set. We report EM of each model before (QS = ✗) and after (QS = ✓) the query-side fine-tuning.

10 Conclusion

In this study, we show that we can learn dense representations of phrases at the Wikipedia scale, which are readily retrievable for open-domain QA and other knowledge-intensive NLP tasks. We learn both phrase and question encoders from the supervision of reading comprehension tasks and introduce two batch-negative techniques to better discriminate phrases at scale. We also introduce query-side fine-tuning that adapts our model to different types of queries. We achieve strong performance on five popular open-domain QA datasets, while reducing the storage footprint and improving latency significantly. We also achieve strong performance on two slot filling datasets using only a small number of training examples, showing the possibility of utilizing our DensePhrases as a knowledge base.

Acknowledgments

We thank Sewon Min, Hyunjae Kim, Gyuwan Kim, Jungsoo Park, Zexuan Zhong, Dan Friedman, Chris Sciavolino for providing valuable comments and feedback. This research was supported by a grant of the Korea Health Technology R&D Project through the Korea Health Industry Development Institute (KHIDI), funded by the Ministry of Health & Welfare, Republic of Korea (grant number: HR20C0021) and National Research Foundation of Korea (NRF-2020R1A2C3010638). It was also partly supported by the James Mi *91 Research Innovation Fund for Data Science and an Amazon Research Award.

Ethical Considerations

Our work builds on standard reading comprehension datasets such as SQuAD to build phrase representations. SQuAD, in particular, is created from a small number of Wikipedia articles sampled from top-10,000 most popular articles (measured by PageRanks), hence some of our models trained only on SQuAD could be easily biased towards the small number of topics that SQuAD contains. We hope that excluding such datasets during training or inventing an alternative pre-training procedure for learning phrase representations could mitigate this problem. Although most of our efforts have been made to reduce the computational complexity of previous phrase retrieval models (further detailed in Appendices A and E), leveraging our phrase retrieval model as a knowledge base will inevitably increase the minimum requirement for the additional experiments. We plan to apply vector quantization techniques to reduce the additional cost of using our model as a KB.

References

Akari Asai, Kazuma Hashimoto, Hannaneh Hajishirzi, Richard Socher, and Caiming Xiong. 2020. Learning to retrieve reasoning paths over wikipedia graph for question answering. In *International Conference on Learning Representations (ICLR)*.

Petr Baudiš and Jan Šedivỳ. 2015. Modeling of the question answering task in the YodaQA system. In *International Conference of the Cross-Language Evaluation Forum for European Languages (CLEF)*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *The Journal of Machine Learning Research (JMLR)*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Qingqing Cao, Harsh Trivedi, Aruna Balasubramanian, and Niranjan Balasubramanian. 2020. Deformer: Decomposing pre-trained Transformers for faster question answering. In *Association for Computational Linguistics (ACL)*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Association for Computational Linguistics (ACL)*.

Danqi Chen and Wen-tau Yih. 2020. Open-domain question answering. In *Association for Computational Linguistics (ACL)*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *JMLR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional Transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Hady Elsahar, Pavlos Vougiouklis, Arslen Remaci, Christophe Gravier, Jonathon Hare, Frederique Laforest, and Elena Simperl. 2018. T-REx: A large scale alignment of natural language with knowledge base triples. In *International Conference on Language Resources and Evaluation (LREC)*.

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. 2010. Building Watson: An overview of the deepqa project. *AI magazine*, 31(3).

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. REALM: Retrieval-augmented language model pre-training. In *International Conference on Machine Learning (ICML)*.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply. *arXiv preprint arXiv:1705.00652*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. 2019. Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *International Conference on Learning Representations (ICLR)*.

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *European Chapter of the Association for Computational Linguistics (EACL)*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734*.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association of Computational Linguistics (TACL)*.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Association for Computational Linguistics (ACL)*.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. *Advances in Neural Information Processing Systems (NIPS)*.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics (TACL)*.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International Conference on Machine Learning (ICML)*.

Jinhyuk Lee, Minjoon Seo, Hannaneh Hajishirzi, and Jaewoo Kang. 2020. Contextualized sparse representations for real-time open-domain question answering. In *Association for Computational Linguistics (ACL)*.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Association for Computational Linguistics (ACL)*.

Kenton Lee, Shimi Salant, Tom Kwiatkowski, Ankur Parikh, Dipanjan Das, and Jonathan Berant. 2017. Learning recurrent span representations for extractive question answering. In *ICLR*.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. Zero-shot relation extraction via reading comprehension. In *Computational Natural Language Learning (CoNLL)*.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2020. Pretrained Transformers for text ranking: BERT and beyond. *arXiv preprint arXiv:2010.06467*.

Lucian Vlad Lita, Abe Ittycheriah, Salim Roukos, and Nanda Kambhatla. 2003. tRuEcasIng. In *Association for Computational Linguistics (ACL)*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NIPS)*.

Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019. A discrete hard EM approach for weakly supervised question answering. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vassilis Plachouras, Tim Rocktäschel, et al. 2021. KILT: a benchmark for knowledge intensive language tasks. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140).

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How much knowledge can you pack into the parameters of a language model? In *Empirical Methods in Natural Language Processing (EMNLP)*.

Minjoon Seo, Tom Kwiatkowski, Ankur Parikh, Ali Farhadi, and Hannaneh Hajishirzi. 2018. Phrase-indexed question answering: A new challenge for scalable document comprehension. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Minjoon Seo, Jinhyuk Lee, Tom Kwiatkowski, Ankur P Parikh, Ali Farhadi, and Hannaneh Hajishirzi. 2019. Real-time open-domain question answering with dense-sparse phrase index. In *Association for Computational Linguistics (ACL)*.

Wissam Siblini, Mohamed Challal, and Charlotte Pasqual. 2020. Delaying interaction layers in transformer-based encoders for efficient open domain question answering. *arXiv preprint arXiv:2010.08422*.

Richard Socher, Christopher D Manning, and Andrew Y Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 deep learning and unsupervised feature learning workshop*.

Ellen M Voorhees et al. 1999. The TREC-8 question answering track report. In *Trec*.

Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. 2019. Multi-passage BERT: A globally normalized BERT model for open-domain question answering. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2019. End-to-end open-domain question answering with bertserini. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Wen-tau Yih, Kristina Toutanova, John C Platt, and Christopher Meek. 2011. Learning discriminative projections for text similarity measures. In *Computational Natural Language Learning (CoNLL)*.

A Computational Cost

We describe the resources and time spent during inference (Table 1 and A.1) and indexing (Table A.1). With our limited GPU resources (24GB $\times$ 4), it takes about 20 hours for indexing the entire phrase representations. We also largely reduced the storage from 1,547GB to 320GB by (1) removing sparse representations and (2) using our sharing and split strategy. See Appendix E for the details on the reduction of storage footprint and Appendix B for the specification of our server for the benchmark.

Indexing	Resources	Storage	Time
DPR	32GB GPU $\times$ 8	76GB	17h
DenSPI + Sparc	24GB GPU $\times$ 4	1,547GB	85h
DensePhrases	24GB GPU $\times$ 4	320GB	20h
Inference	RAM / GPU	#Q/sec (GPU, CPU)	
DPR	86GB / 17GB	0.9, 0.04	
DenSPI + Sparc	27GB / 2GB	2.1, 1.7	
DensePhrases	12GB / 2GB	20.6, 13.6	

Table A.1: Complexity analysis of three open-domain QA models during indexing and inference. For inference, we also report the minimum requirement of RAM and GPU memory for running each model with GPU. For computing #Q/s for CPU, we do not use GPUs but load all models on the RAM.

B Server Specifications for Benchmark

To compare the complexity of open-domain QA models, we install all models in Table 1 on the same server using their public open-source code. Our server has the following specifications:

Hardware
Intel Xeon CPU E5-2630 v4 @ 2.20GHz
128GB RAM
12GB GPU (TITAN Xp) $\times$ 2
2TB 970 EVO Plus NVMe M.2 SSD $\times$ 1

Table B.2: Server specification for the benchmark

For DPR, due to its large memory consumption, we use a similar server with a 24GB GPU (TITAN RTX). For all models, we use 1,000 randomly sampled questions from the Natural Questions development set for the speed benchmark and measure #Q/sec. We set the batch size to 64 for all models except BERTSerini, ORQA and REALM, which do not allow a batch size of more than 1 in their open-source implementations. #Q/sec for DPR includes retrieving passages and running a reader model and the batch size for the reader model is set to 8 to fit in the 24GB GPU (retriever batch size is still 64). For other hyperparameters, we use the default settings of each model. We also exclude the time and the number of questions in the first five iterations for warming up each model. Note that despite our effort to match the environment of each model, their latency can be affected by various different settings in their implementations such as the choice of library (PyTorch vs. Tensorflow).

Dataset	**Train**	**Dev**	**Test**
Natural Questions	79,168	8,757	3,610
WebQuestions	3,417	361	2,032
CuratedTrec	1,353	133	694
TriviaQA	78,785	8,837	11,313
SQuAD	78,713	8,886	10,570
T-REx	2,284,168	5,000	5,000
Zero-Shot RE	147,909	3,724	4,966

Table C.3: Statistics of five open-domain QA datasets and two slot filling datasets. We follow the same splits in open-domain QA for the two reading comprehension datasets (SQuAD and Natural Questions).

C Data Statistics and Pre-processing

In Table C.3, we show the statistics of five open-domain QA datasets and two slot filling datasets. Pre-processed open-domain QA datasets are provided by Chen et al. (2017) except Natural Questions and TriviaQA. We use a version of Natural Questions and TriviaQA provided by Min et al. (2019); Lee et al. (2019), which are pre-processed for the open-domain QA setting. Slot filling datasets are provided by Petroni et al. (2021). We use two reading comprehension datasets (SQuAD and Natural Questions) for training our model on Eq. (9). For SQuAD, we use the original dataset provided by the authors (Rajpurkar et al., 2016). For Natural Questions (Kwiatkowski et al., 2019), we use the pre-processed version provided by Asai et al. (2020).[9] We use the short answer as a ground truth answer a^* and its long answer as a gold passage p. We also match the gold passages in Natural Questions to the paragraphs in Wikipedia whenever possible. Since we want to check the performance changes of our model with the growing number of tokens, we follow the same split (train/dev/test) used in Natural Questions-Open for the reading comprehension setting as well. During the valida-

[9] https://github.com/AkariAsai/learning_to_retrieve_reasoning_paths

tion of our model and baseline models, we exclude samples whose answers lie in a list or a table from a Wikipedia article.

D Hyperparameters

We use the Adam optimizer (Kingma and Ba, 2015) in all our experiments. For training our phrase and question encoders with Eq. (9), we use a learning rate of 3e-5 and the norm of the gradient is clipped at 1. We use a batch size of B =84 and train each model for 4 epochs for all datasets, where the loss of pre-batch negatives is applied in the last two epochs. We use SQuAD to train our QG model[10] and use spaCy[11] for extracting named entities in each training passage, which are used to generate questions. The number of generated questions is 327,302 and 1,126,354 for SQuAD and Natural Questions, respectively. The number of preceding batches C is set to 2.

For the query-side fine-tuning with Eq. (11), we use a learning rate of 3e-5 and the norm of the gradient is clipped at 1. We use a batch size of 12 and train each model for 10 epochs for all datasets. The top k for the Eq. (11) is set to 100. While we use a single 24GB GPU (TITAN RTX) for training the phrase encoders with Eq. (9), query-side fine-tuning is relatively cheap and uses a single 12GB GPU (TITAN Xp). Using the development set, we select the best performing model (based on EM) for each dataset, which are then evaluated on each test set. Since SpanBERT only supports cased models, we also truecase the questions (Lita et al., 2003) that are originally provided in the lowercase (Natural Questions and WebQuestions).

E Reducing Storage Footprint

As shown in Table 1, we have reduced the storage footprint from 1,547GB (Lee et al., 2020) to 320GB. We detail how we can reduce the storage footprint in addition to the several techniques introduced by Seo et al. (2019).

First, following Seo et al. (2019), we apply a linear transformation on the passage token representations to obtain a set of filter logits, which can be used to filter many token representations from $\mathcal{W}(\mathcal{D})$. This filter layer is supervised by applying the binary cross entropy with the gold start/end positions (trained together with Eq. (9)). We tune the threshold for the filter logits on the reading comprehension development set to the point where the performance does not drop significantly while maximally filtering tokens. In the full Wikipedia setting, we filter about 75% of tokens and store 770M token representations.

Second, in our architecture, we use a base model (SpanBERT-base) for a smaller dimension of token representations ($d = 768$) and does not use any sparse representations including tf-idf or contextualized sparse representations (Lee et al., 2020). We also use the scalar quantization for storing `float32` vectors as `int4` during indexing.

Lastly, since the inference in Eq. (10) is purely based on MIPS, we do not have to keep the original start and end vectors which takes about 500GB. However, when we perform query-side fine-tuning, we need the original start and end vectors for reconstructing them to compute Eq. (11) since (the on-disk version of) MIPS index only returns the top-k scores and their indices, but not the vectors.

[10] The quality of generated questions from a QG model trained on Natural Questions is worse due to the ambiguity of information-seeking questions.

[11] https://spacy.io/